Gamelet: A Mobile Service Component for Building Multi-server DVE on Grid

Tianqi Wang, Cho-Li Wang, Francis C.M.Lau
Department of Computer Science
The University of Hong Kong
What is DVE?

A Distributed Virtual Environment (DVE) system is a software system through which people who are geographically dispersed over the world can interact with each other by sharing a consistent environment in terms of space, presence and time.

- realistic 3D graphics
- real-time interaction
- a large number of users
Many DVE Applications

- Military team training
- Collaborative design and engineering
- Increasingly used for:
  - virtual shopping mall
  - Interactive e-learning
  - Multiplayer online games
General Client-server Model

**Server Tasks:**
- Receive user messages
- Calculate world state
- Ensure objects consistency
- Message filtering/compressing
- Administration work

**Client Tasks:**
- Scene rendering
- Simple calculations
What are the problems

- **World State Consistency**
- **Real-time Response**

- The number of users is unpredictable.
  - Support scalability and dynamic resource aggregation

- Workload imbalance among servers as users may act freely in the virtual world
  - Need dynamic load transfer support
Grid Computing

- **Grid Computing** (I. Foster)
  - Concerned with flexible, secure, coordinated resource sharing among dynamic collections of virtual organizations

- **Grid Service:**
  - A kind of stateful, transient web service
  - Large-scale scientific experiments (DOE Science Grid)
  - Large-scale data analysis (EU Data Grid)
DVE Systems on Grid

- **Butterfly Grid**
  - Easy to use **commercial** grid computing environment for developers
  - High-performance networked servers for the publishers

- **Cal-(IT)$^2$ Game Grid**
  - Provides the first massively multi-user online game grid for research, teaching, art, and experimentation
Challenges

- How to re-design the existing DVE system into an open and service oriented system
  - Map monopolistic model into OGSA framework

- How to ensure the quality of service from the end users’ perspective
  - Dynamic load migration/balancing
Our Approach

- Propose a service-oriented framework
  - Based on a component called *gamelet*

- Propose an *Adaptive Gamelet Load-balancing (AGL)* algorithm
Multi-server Architecture

- **Layered design:**
  - Monitor Server
  - Worker Server
  - Communicator Server

![Diagram showing layered design with Monitor Layer, Gamelet Layer, and Communicator Layer.]
Gamelet Concept

- A mobile service component that is responsible for processing the workload introduced by a partitioned virtual environment.

- Discussions:
  - Load awareness
  - High mobility
  - Embedded Synchronization
System Framework

Monitor

Gamelet Wrapper

Gamelet Wrapper

Synchronization & Migration Message

Service Container

SOAP

World Management
Collision Detection Management

Gamelet Factory Services

GRAM

Index Service

GSI

NS/ Life Cycle Management

Service Container

Synchronization Management

Avatar Management

Communicator

Partition Management

Client

Update

World Management

Collision Detection Management

Synchronization Management

Avatar Management

Gamelet Factory Services

GRAM

Index Service

GSI

NS/ Life Cycle Management

Service Container

Client

Client

Client
Message Route/Gamelet Migration

![Diagram of message route and gamelet migration](image)

- Client
- Communicator
- Server A
- Server B
- Monitor
- Command Packets
- Updates
- Workload Parameters
- Migration Control
- Post Migration
- Time

:Gamelet
: Gamelet Factory
: Message Queue
Load Balancing Strategy

- Special characteristics of Grid environment:
  - High latency, heterogeneous machines, etc

- *Adaptive Gamelet Load-balancing* algorithm:
  - Use more accurate workload model
  - Adapt to the network latency and resource heterogeneity
AGL Algorithm

- Graph repartition problem

1. For each gamelet $G_i$, create a node $N_i$ in the graph $G$
2. For any two nodes $N_i$ and $N_j$, if there are some inter-communications $C_{i,j}$ between them, create an edge between $N_i$ and $N_j$ with value $W_{i,j} = C_{i,j}$.
Threshold delta

AGL Algorithm

1. **Select server** \( m, n \)

2. **Cost**\((G_i, n) = Syn'(G_i) - Syn(G_i)\)
   
   \(Syn(G_i) = \text{Sum}\{W(i,j) \times \text{Latency}(m,n)\}\)

3. **Select a gamelet with the smallest Cost**\((G_i, n)\)

4. **Estimate workload transferred:**
   
   \(\text{Percentage}(G_i) = \frac{\text{Val}(G_i)}{\text{Sum}\{\text{Val}(G_j)\}}\)
   
   \(\text{Val}(G_i) : \text{weighted package sending rate}\)

5. **Do 1-4 until original server is under threshold**

6. **If there is still an overloaded server, add a new Grid server, go to step 1.**
Twofold Meaning

- Adapt to the network latency
- Cost model: synchronization cost and grid inter-server latency
- Workload evaluation based on the activities of the clients and also consider the resource heterogeneity
Prototype Design and Implementation

- Virtual environment:
  - Partitioned world: 100*100*20
  - Overlapping length: 5

- Client simulator
  - Random movement per 100ms
  - Hotspot (25ms)
  - Data packets (32 B)

- Performance parameters
  - Response time
  - System Capacity
Testing Environment

- Gamelet and Monitor (GT3)
- Client simulator, communicator (J2SE 1.4.2)
- Server configurations
  - Linux kernel 2.4.18, P4 2.0GHz CPU
  - 512M RAM, 100Mbps Ethernet
  - Default latency within several million seconds

<table>
<thead>
<tr>
<th>Latency (ms)</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>100</td>
<td>1</td>
<td>200</td>
<td>150</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>50</td>
</tr>
</tbody>
</table>
Gamelet Creation and Migration

- Monitor creates 16 gamelets sequentially on 1/2/4 servers respectively

- Gamelet migration time
  - The number of avatars
  - Interaction overhead: 30 – 40 ms
AGL Algorithm Evaluation

- Delta = 90%;
- Initially, 16 gamelets in one server; servers are added as necessary
- Compare with even-avatar algorithm (EAL)
Average Response Time

Figure 1

<table>
<thead>
<tr>
<th>91 Clients</th>
<th>CPU Load</th>
<th>Inter-server Traffic</th>
<th>RT (ms)</th>
<th>ART (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Servers</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
</tr>
<tr>
<td>AGL</td>
<td>90%</td>
<td>89%</td>
<td>79%</td>
<td>81%</td>
</tr>
<tr>
<td>EAL</td>
<td>100%</td>
<td>100%</td>
<td>99%</td>
<td>42%</td>
</tr>
</tbody>
</table>

Figure 2

<table>
<thead>
<tr>
<th>176 Clients</th>
<th>CPU Load</th>
<th>Inter-server Traffic</th>
<th>RT (ms)</th>
<th>ART (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Servers</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
</tr>
<tr>
<td>AGL</td>
<td>90%</td>
<td>72%</td>
<td>69%</td>
<td>71%</td>
</tr>
<tr>
<td>EAL</td>
<td>90%</td>
<td>88%</td>
<td>89%</td>
<td>91%</td>
</tr>
</tbody>
</table>

Figure 3

- Influence of network latency
- Influence of workload model
System Capacity

- More scalable and cost-effective
- Increase by up to 80%
Related Work

- **CittaTron** (Osaka University, 2001)
  - Multi-server networked Internet game
  - Only consider user number for load transfer

- **Cyber-walk** (CTU, 2002)
  - A distributed web walk through system
  - Partition is adjustable
  - Several hotspots -> cascading effect

- **NetEffect** (NUS, 1997)
  - VE divided into separated communities
  - Non-transparent load balancing

In our proposed gamelet-based multi-server framework, these problem are resolved.
Thank You!
Questions?